Department Informatik
Lehrstuhl für Informatik 3 (Rechnerarchitektur)
Friedrich-Alexander-Universität Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen
11.02.2013
Klausur zu

Grundlagen der Rechnerarchitektur und -organisation

.....	.....	.....	.....
Matrikelnummer	Geburtsdatum	Vorname	Name

- Es sind *keine* Hilfsmittel erlaubt!
- Legen Sie den Ausweis (mit Lichtbild!) griffbereit auf den Platz!
- Dieses Aufgabenheft umfasst 20 Seiten (ohne Anhang). Überprüfen Sie die Vollständigkeit!
- Gesondert beigelegte Blätter werden nicht bewertet.
- Schreiben Sie deutlich! Unleserliches wird nicht bewertet!
- Es darf nicht mit der Farbe rot geschrieben werden!
- Offensichtlich falsche oder überflüssige Antworten können zu Punktabzug führen!
- Begründen Sie Ihre Antworten!

Durch meine Unterschrift bestätige ich

- den Empfang der vollständigen Klausurunterlagen
- die Kenntnisnahme der obigen Informationen.

Erlangen, den 11.02.2013 .....
(Unterschrift)

Ich bin damit einverstanden, dass mein Prüfungsergebnis der Klausur unter Angabe der Matrikelnummer veröffentlicht wird.

Erlangen, den 11.02.2013 .....
(Unterschrift)

Aufgabe	1	2	3	4	5	6	7
max. Punktzahl	13	13	15	10	15	12	12
erreichte Punktzahl							

Summe	/90		
Bonus	/15		
Gesamt		Note	

Aufgabe 1: Assemblerprogrammierung — 13 Punkte

Gegeben sei das folgende Programm:

```

calls.o:     file format elf32-i386


Disassembly of section .text:

00000000 <a>:
   0:   e8 0b 00 00 00          call   10 <b>
   5:   6a 42                   push   $0x42
   7:   e8 05 00 00 00          call   11 <c>
   c:   83 c4 04                add    $0x4,%esp
   f:   c3                      ret

00000010 <b>:
  10:   c3                      ret

00000011 <c>:
  11:   c3                      ret

00000012 <main>:
  12:   e8 e9 ff ff ff          call   0 <a>
  17:   6a 43                   push   $0x43
  19:   e8 f3 ff ff ff          call   11 <c>
  1e:   83 c4 04                add    $0x4,%esp
  21:   b8 00 00 00 00          mov    $0x0,%eax
  26:   c3                      ret
```

Ein Befehl ist wie folgt aufgebaut (AT&T-Syntax):

Adresse (hex)	Maschinenbefehl	Assemblerbefehl	Operanden	Bedeutung
`c:`	`83 c4 04`	`add`	`$0x4 ,%esp`	`esp=esp+0x4`

Hinweis: Am Ende der Klausur finden Sie eine Kopie der Angabe, die Sie heraustrennen dürfen.

1. Führen Sie die Funktion <main> aus. Skizzieren Sie dabei den Aufbau des Stacks jeweils nach Ausführung einer `call`-Instruktion, also bevor die erste Instruktion der aufgerufenen Funktion ausgeführt wurde!

   Geben Sie jeweils auch den aktuellen Stackpointer an!

   Der Stack wachse in Richtung der Adresse `0x00 00 00 00`. Der Stackpointer `%esp` zeige immer auf den Anfang des letzten Eintrags und hat zu Beginn den Wert `0xff ff d2 cc`.

   Alle Adressen und Daten auf dem Stack sind 32 Bit breit. 8 Punkte

2. Ordnen Sie den folgenden Programmfragmenten a) – d) *alle* in Frage kommenden Befehlssatzarchitektur-Klassen zu, und unterstreichen Sie mindestens eine für Ihre Zuordnung ausschlaggebende Instruktion. Eine weitere Erklärung ist nicht notwendig!

   `R1`, `R2` sind Register, `A-C` sind Variablen im Speicher, `#` kennzeichnet unmittelbar adressierte Operanden.

   Beispiel: `add R1, C` addiert den Inhalt der Variablen `C` auf das Register `R1`. 5 Punkte

   a)
   ```
   xor R1, R2
   xor R2, R1
   xor R1, R2
   ```

   b)
   ```
   push #5
   push C
   sub
   pop A
   ```

   c)
   ```
   push R1
   ld R1, A
   add R1, B
   st R1, C
   pop R1
   ```

   d)
   ```
   ld C
   add #2
   st C
   ```

Aufgabe 2: Mikroprogrammierung — 13 Punkte

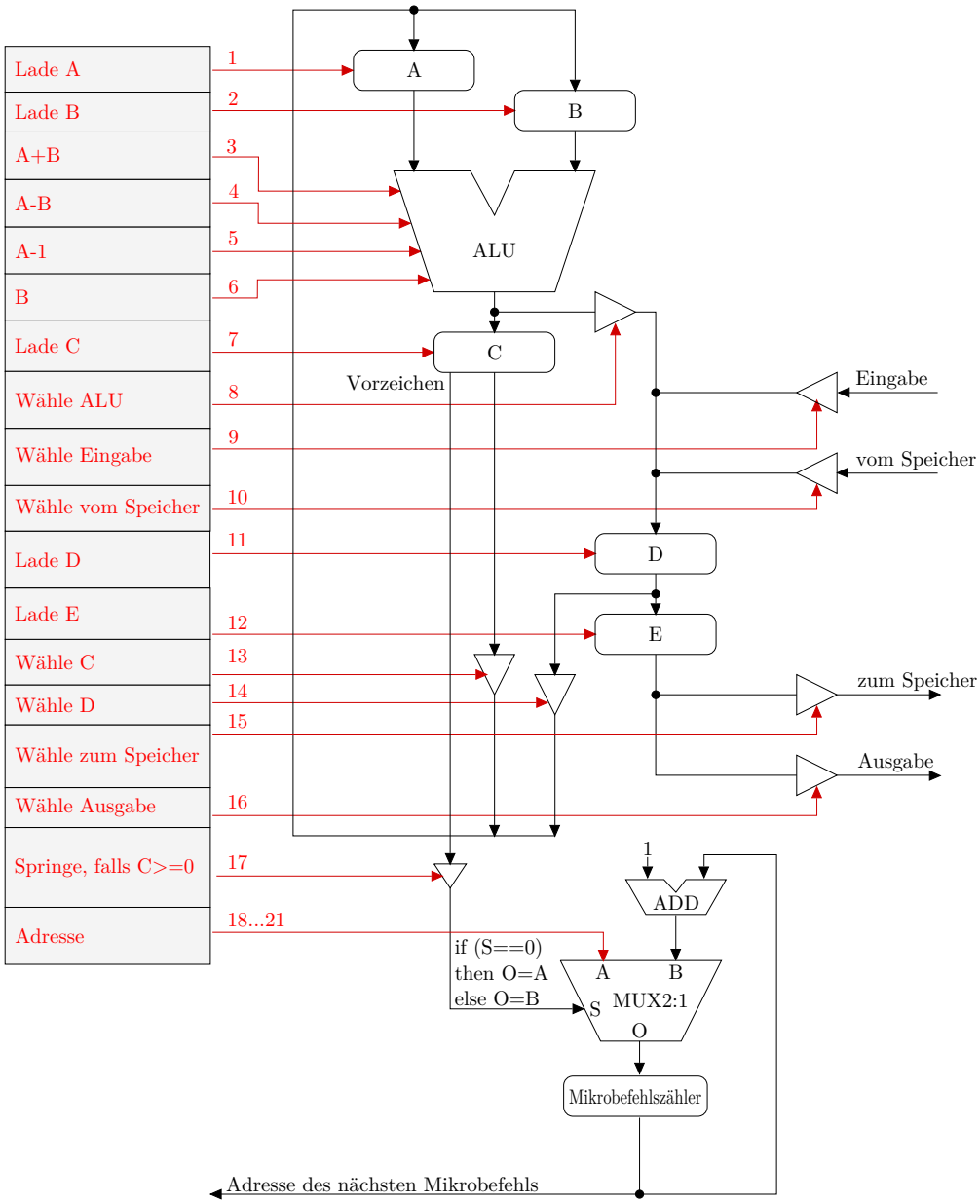
1. Wozu wird Mikroprogrammierung in der CPU eingesetzt? — 1 Punkt

2. Setzen Sie die Begriffe *Mikrobefehl* und *Makrobefehl* in Bezug zueinander! — 1 Punkt

3. Welche Alternative zur Mikroprogrammierung gibt es? — 1 Punkt

4. Nennen Sie jeweils einen Vorteil beider Techniken! — 2 Punkte

Gegeben sei der folgende Teil einer CPU:



Hinweis: Am Ende der Klausur finden Sie eine Kopie der Angabe, die Sie heraustrennen dürfen.

5. Schreiben Sie ein Mikroprogramm für obige CPU, das eine natürliche Zahl n von der Eingabe liest, und die Summe aller natürlichen Zahlen von 0 bis n auf der Ausgabe zurückgibt.

   Verwenden Sie die folgende Tabelle. Leer gelassene Steuerleitungen entsprechen dem Wert „0", bei Sprüngen muss die Zieladresse explizit angegeben werden! Das niederwertigste Bit des Sprungziels entspricht Steuerleitung 21. Alle Register haben zu Beginn den Wert 0.

   Hinweis:
   Überlegen Sie sich zuerst den Ablauf in Pseudocode. 8 Punkte

(Die Tabellenlänge entspricht nicht der erwarteten Mikroprogrammlänge!)

Adr.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	Kommentar
0																						
1																						
2																						
3																						
4																						
5																						
6																						
7																						
8																						
9																						
10																						
11																						
12																						
13																						
14																						
15																						

Aufgabe 3: Speicherverwaltung — 15 Punkte

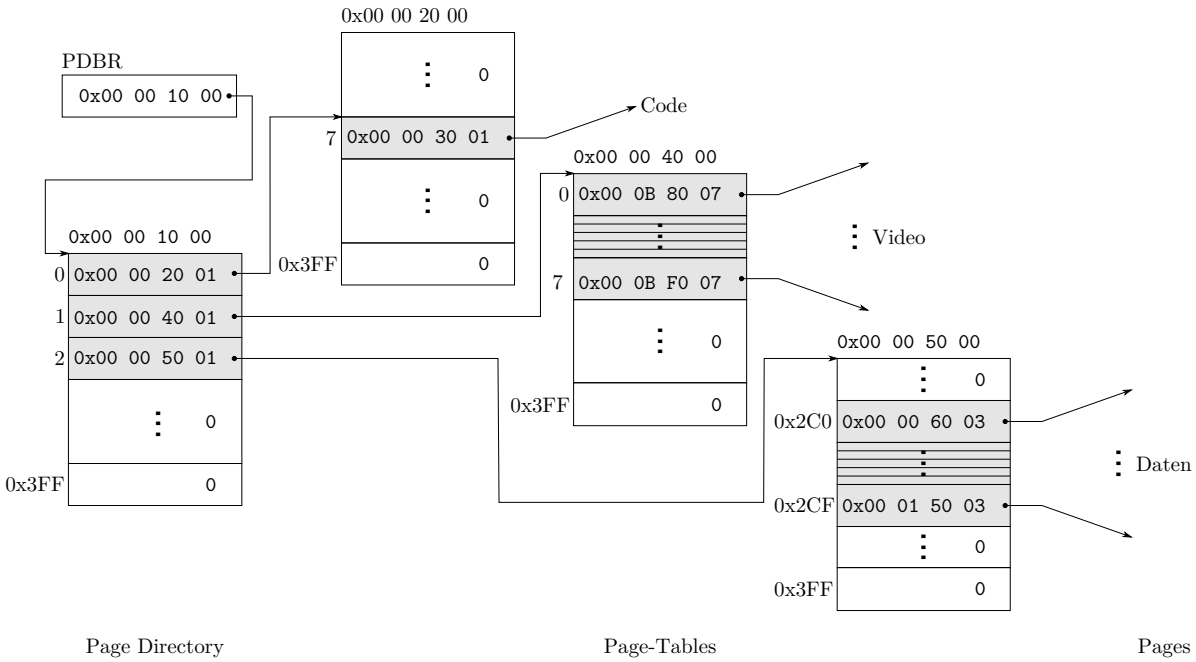
Eine CPU biete eine Memory-Management-Unit mit folgenden Eigenschaften:

- zweistufige Adresstabellen
- je 1024 Einträge zu je 4 Byte in den Tabellen
  - Bit 31-12: höherwertige Bits der physikalischen Adresse
  - Bit 11-3: unbenutzt
  - Bit 2: Cache-Disabled-Bit
  - Bit 1: Write-Enable-Bit
  - Bit 0: Present-Bit
- Pages zu je 4 KiB Größe

Folgendes ist für ein Programm erfüllt:

- im virtuellen Adressbereich `0x00 00 70 00` bis `0x00 00 7F FF` steht schreibgeschützter Code,
- ab der virtuellen Adresse `0x00 40 00 00` ist der Video Memory der VGA-Karte eingeblendet,
- im virtuellen Adressbereich `0x00 AC 00 00` bis `0x00 AC FF FF` sind die Daten gespeichert,
- alle anderen Bereiche lösen Zugriffsfehler aus,
- die physikalischen Bereiche liegen sequentiell im Speicher.

Schematisch sieht der Adressraum wie folgt aus:



Hinweis: Am Ende der Klausur finden Sie eine Kopie der Angabe, die Sie heraustrennen dürfen.

1. Welche Einträge stehen in einem vollassoziativen TLB mit vier Einträgen, nachdem die CPU nacheinander auf die virtuellen Adressen `0x00 00 71 2C`, `0x00 40 31 F0`, `0x00 AC C0 F0` und `0x00 00 71 30` lesend zugegriffen hat? Skizzieren Sie hierfür den TLB!

   Der TLB sei zu Beginn leer. Ignorieren Sie dabei die Ersetzungsstrategie!

   5 Punkte

2. Was passiert, wenn die CPU auf die virtuelle Adresse `0x00 00 68 88` lesend zugreift?

   Begründen Sie Ihre Antwort! 1 Punkt

3. Was muss beim Prozesswechsel mit der Pagetabelle und dem TLB gemacht werden?

   2 Punkte

4. Warum wird meist mehrstufiges Paging statt einer einzelnen Pagetabelle verwendet?

   Belegen Sie Ihre Aussage rechnerisch anhand einer 32-Bit Architektur mit 4 KiB großen Pages!
   5 Punkte

5. Wie könnte bei Segmentierung ein gemeinsamer Arbeitsspeicherbereich zwischen mehreren Prozessen realisiert werden? Ist dies auch bei Paging möglich? 2 Punkte

Aufgabe 4: Umformung — 10 Punkte

Gegeben sei folgendes Hochsprachenprogramm:

```
for (i = -LIMIT; i <= LIMIT; i++) {
  if (i == 0 || pos+i < 0) {
    continue;
  } else if (pos+i > LENGTH-1) {
    break;
  }
  ids[pos+i]--;
}
```

1. Formen Sie das Programm so in if-goto-Darstellung um, dass es sich möglichst leicht in Assembler übersetzen lässt. Reduzieren Sie dazu auch alle arithmetischen und logischen Ausdrücke auf maximal zwei Operanden. Falls nötig, fügen Sie temporäre Variablen ein.

   Verändern Sie die eigentlichen Operationen nicht! Arrayzugriffe selbst müssen nicht transformiert werden, ggf. lediglich die Indexberechnung. 8 Punkte

2. Das Konstrukt `switch...case` kann analog zu `if...else if...else` auf Assemblerinstruktionen abgebildet werden.

   Warum kann dadurch die Laufzeit eines Programms sehr hoch werden? 2 Punkte

Aufgabe 5: Arbeitsspeicher — 15 Punkte

1. Was unterscheidet dynamischen RAM (DRAM) von statischem RAM? Worin liegt jeweils der Nachteil? 2 Punkte

2. Warum sollten aufeinanderfolgende Datenworte auf unterschiedlichen Speicherbänken abgelegt werden? Wie nennt sich dieses Vorgehen? 2 Punkte

3. Beschreiben Sie (in Stichpunkten), was jeweils zwischen zwei aufeinanderfolgenden Speichergenerationen seit einfachem SDRAM (SDRAM bis DDR3-SDRAM) geändert wurde und wie sich dadurch der Durchsatz geändert hat!

   Wie hat sich der Speichertakt verändert? 7 Punkte

4. Warum ist es vom Speicherzugriffsmuster abhängig, ob ein Programm auf einem Rechner mit DDR-SDRAM statt einfachem SDRAM schneller abgearbeitet wird, sofern kein Cache verwendet wird? 2 Punkte

5. Warum führt die Verwendung von DDR-SDRAM statt SDRAM in Kombination mit einem Daten-Cache mit 64 Byte Blockgröße unabhängig von Speicherzugriffsmuster und Cache-Hit-Rate zu einer Beschleunigung der Programmabarbeitung? 2 Punkte

Aufgabe 6: Parallelverarbeitung 12 Punkte

1. Was versteht man unter Pipelining von Befehlen in der CPU? 2 Punkte

2. Nennen Sie drei verschiedenen Arten von Multithreading auf einer CPU und geben Sie an, wieviele Threadkontexte (Befehlszähler, Universalregistersätze, Flags-Register) jeweils gleichzeitig aktiv sind. 3 Punkte

3. Gegeben sei eine Architektur mit einer sechsstufigen Pipeline, bestehend aus *Befehl holen* (BH), *Befehl decodieren*, (BD) *Operanden holen*, (OH) *Befehl ausführen (BA1, BA2)*, *Ergebnis sichern* (ES):

BH BD OH BA1 BA2 ES

Die Pipeline verfüge über keine erweiterten Mechanismen wie Sprungvorhersage, spekulative Ausführung, Forwarding, etc. Um dennoch ein korrektes Ergebnis zu garantieren, soll statt dessen die Ausführung weiterer Instruktionen so lange verzögert werden, bis kein Konflikt mehr vorliegt, allerdings nur, falls tatsächlich notwendig.

Bei Sprungbefehlen soll der Befehlszähler so früh wie möglich aktualisiert werden. Jeder Befehl muss die gesamte Pipeline durchlaufen.

Führen Sie das folgende Programm aus:

```
1    xorl %eax, %eax
2    movl $1, %ecx
3  .Lbody:
4    addl %ecx, %eax        -- eax = eax+ecx
5    subl $1, %ecx
6  .Lcond:
7    cmpl $0, %ecx
8    jg .Lbody              -- springe, falls %ecx > 0
9    xorl $3, %ecx
```

> **Hinweis:** Am Ende der Klausur finden Sie eine Kopie der Angabe, die Sie heraustrennen dürfen.

Tragen Sie in der Tabelle auf der nächsten Seite in jeder Stufe *die Zeile* des dort gerade ausgeführten Befehls ein. Geben Sie außerdem den aktuellen Registerinhalt von `eax` und `ecx` an, soweit bekannt.

Der Befehl `cmpl` aktualisiert das Statusregister zur Sprungauswertung erst in der Phase *ES*, `jg` wertet es in der *OH*-Phase aus.

Lassen Sie am Ende die Pipeline leer laufen.

Die Tabellenlänge entspricht nicht der erwarteten Ausführungszeit! 7 Punkte

Takt	BH	BD	OH	BA1	BA2	ES	`eax`	`ecx`
0								
1								
2								
3								
4								
5								
6								
7								
8								
9								
10								
11								
12								
13								
14								
15								
16								
17								
18								
19								
20								
21								
22								
23								
24								
25								
26								
27								
28								
29								
30								
31								
32								
33								
34								

Aufgabe 7: Cache — 12 Punkte

Eine 32-Bit CPU verwende einen direktabbildenden Daten-Cache mit insgesamt 512 Blöcken. Jeder Block umfasse 8 Bytes. Der Cache sei zu Beginn leer.

Folgende Funktion soll ausgeführt werden:

```
1  #define SIZE 1024
2  //sizeof(int) = 4
3
4  void vadd(int *a, int *b, int *c) {
5    int i;
6    for (i = 0; i < SIZE; i++) {
7      c[i] = a[i] + b[i];
8    }
9    return;
10 }
```

Die Arrays `a,b` und `c` umfassen jeweils `SIZE` Elemente und liegen ab Adresse `0x00 10 00 00` sequentiell im Speicher. Die Laufvariable `i` befindet sich in einem Register. Variablen vom Typ `int` sind 4 Byte breit.

1. Geben Sie zu jedem der drei Arrays den Adressbereich im Speicher an! 3 Punkte

2. Berechnen Sie den Index der Cache-Blöcke, an denen die Werte `a[0]`,`b[0]` und `c[0]` abgelegt werden! 1 Punkt

3. Pro Block können zwei Integer-Werte gespeichert werden. Warum wird die Hit-Rate des Daten-Caches bei den obigen Array-Zugriffen trotzdem deutlich unter 50% liegen? 2 Punkte

Ein Cache besitzt folgende Adressaufteilung:

31 ... 16	15 ... 6	5 ... 0
Tag	Index	Byteadr

Der Cache bietet Platz für 256 KiB Nutzdaten. Der Speicher ist byteweise adressierbar.

4. Bestimmen Sie die Größe in Byte eines Cache-Blocks! 1 Punkt

5. Wieviele Blöcke umfasst der Cache? 1 Punkt

6. Welche Organisationsform verwendet der Cache?

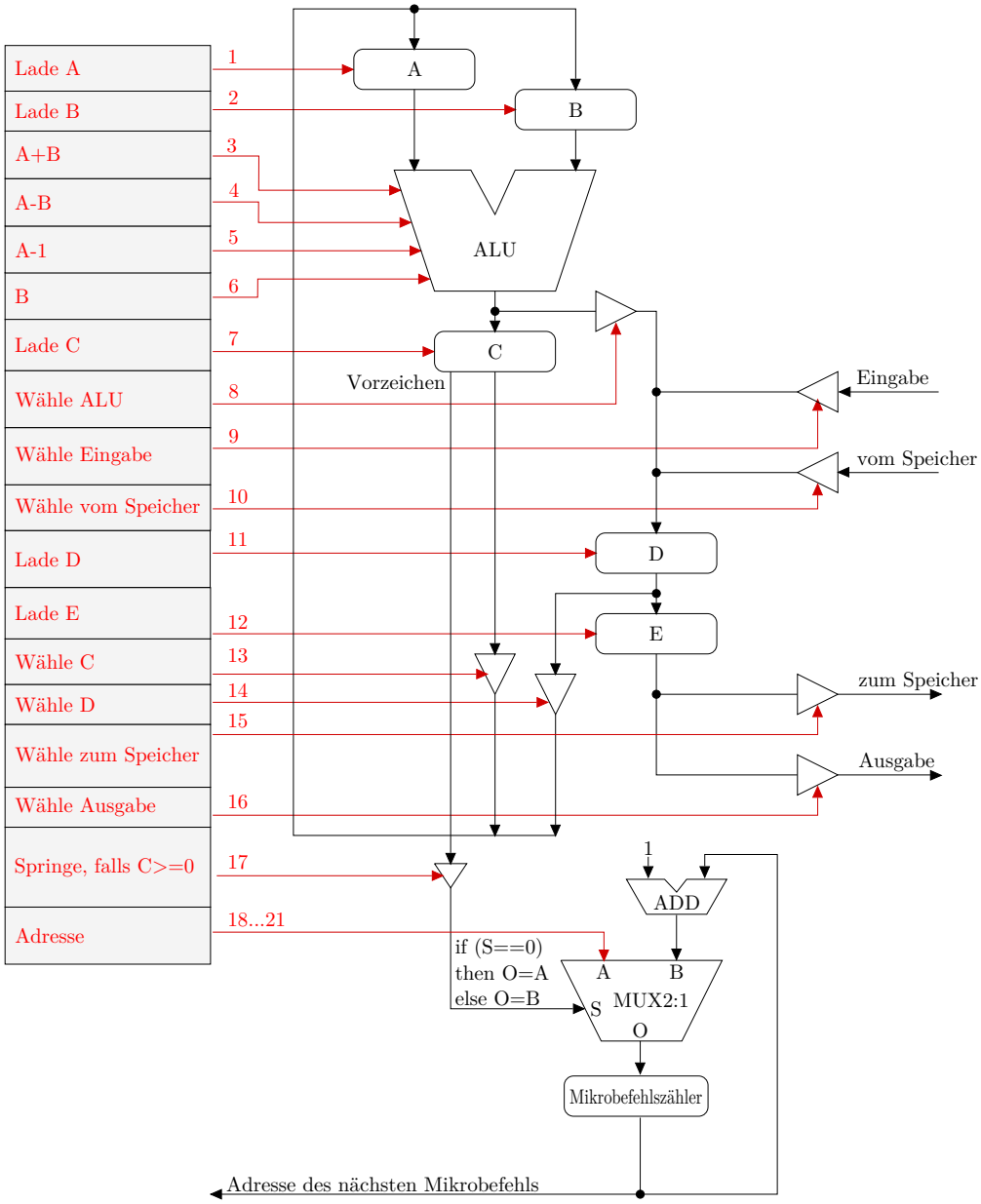
   Begründen Sie Ihre Antwort! 4 Punkte

Zusätzlicher Platz

```
1
2  calls.o:     file format elf32-i386
3
4
5  Disassembly of section .text:
6
7  00000000 <a>:
8     0:   e8 0b 00 00 00          call   10 <b>
9     5:   6a 42                   push   $0x42
10    7:   e8 05 00 00 00          call   11 <c>
11    c:   83 c4 04                add    $0x4,%esp
12    f:   c3                      ret
13
14 00000010 <b>:
15   10:   c3                      ret
16
17 00000011 <c>:
18   11:   c3                      ret
19
20 00000012 <main>:
21   12:   e8 e9 ff ff ff          call   0 <a>
22   17:   6a 43                   push   $0x43
23   19:   e8 f3 ff ff ff          call   11 <c>
24   1e:   83 c4 04                add    $0x4,%esp
25   21:   b8 00 00 00 00          mov    $0x0,%eax
26   26:   c3                      ret
```

Ein Befehl ist wie folgt aufgebaut (AT&T-Syntax):

Adresse (hex)	Maschinenbefehl	Assemblerbefehl	Operanden	Bedeutung
`c:`	`83 c4 04`	`add`	`$0x4 ,%esp`	`esp=esp+0x4`


Lade A
Lade B
A+B
A-B
A-1
B
Lade C
Wähle ALU
Wähle Eingabe
Wähle vom Speicher
Lade D
Lade E
Wähle C
Wähle D
Wähle zum Speicher
Wähle Ausgabe
Springe, falls C>=0
Adresse
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18...21
A
B
ALU
C
Vorzeichen
Eingabe
vom Speicher
D
E
zum Speicher
Ausgabe
1
ADD
if (S==0)
then O=A
else O=B
A
B
S
MUX2:1
O
Mikrobefehlszähler
Adresse des nächsten Mikrobefehls

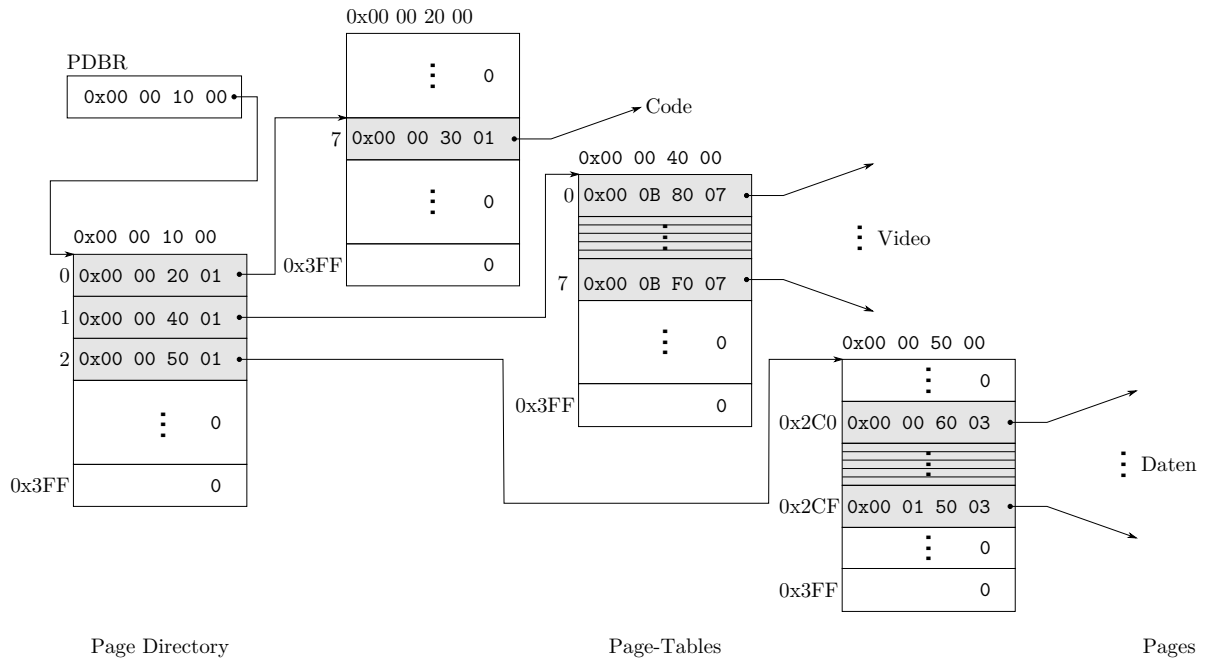
Eine CPU biete eine Memory-Management-Unit mit folgenden Eigenschaften:

- zweistufige Adresstabellen
- je 1024 Einträge zu je 4 Byte in den Tabellen
  - Bit 31-12: höherwertige Bits der physikalischen Adresse
  - Bit 11-3: unbenutzt
  - Bit 2: Cache-Disabled-Bit
  - Bit 1: Write-Enable-Bit
  - Bit 0: Present-Bit
- Pages zu je 4 KiB Größe

Folgendes ist für ein Programm erfüllt:

- im virtuellen Adressbereich `0x00 00 70 00` bis `0x00 00 7F FF` steht schreibgeschützter Code,
- ab der virtuellen Adresse `0x00 40 00 00` ist der Video Memory der VGA-Karte eingeblendet,
- im virtuellen Adressbereich `0x00 AC 00 00` bis `0x00 AC FF FF` sind die Daten gespeichert,
- alle anderen Bereiche lösen Zugriffsfehler aus,
- die physikalischen Bereiche liegen sequentiell im Speicher.

Schematisch sieht der Adressraum wie folgt aus:



```
1   xorl %eax, %eax
2   movl $1, %ecx
3 .Lbody:
4   addl %ecx, %eax        -- eax = eax+ecx
5   subl $1, %ecx
6 .Lcond:
7   cmpl $0, %ecx
8   jg .Lbody              -- springe, falls %ecx > 0
9   xorl $3, %ecx
```